

Evaluating functions as processes

Beniamino Accattoli

Carnegie Mellon University - Pittsburgh, PA, US

A famous result by Milner is that the λ -calculus can be simulated inside the π -calculus. This simulation, however, holds only modulo strong bisimilarity on processes, i.e. there is a slight mismatch between β -reduction and how it is simulated in the π -calculus. The idea is that evaluating a λ -term in the π -calculus is like running an environment-based abstract machine, rather than applying ordinary β -reduction. In this paper we show that such an abstract-machine evaluation corresponds to linear weak head reduction, a strategy arising from the representation of λ -terms as linear logic proof nets, and that the relation between the two is as tight as it can be. The study is also smoothly rephrased in the call-by-value case, introducing a call-by-value analogous of linear weak head reduction.

Introduction

A key result about the expressiveness of the π -calculus is that it can represent the λ -calculus, as it has been showed by Robin Milner [33]. During the nineties the relationship between the two systems has been explored in-depth, mostly by Davide Sangiorgi [36, 37] and Gérard Boudol [14, 13]. Nowadays, it takes a relevant part in the standard reference for the π -calculus [38], and in any introductory course about it. From the process calculus point of view, it helps in getting deeper insights into its theory, especially because the π -calculus is far less canonical than the λ -calculus. From the λ -calculus point of view, it provides new tools to analyze the behavior of λ -terms and the dynamics of β -reduction.

The idea is that the π -calculus can be considered as a sort of flexible abstract machine to which the λ -calculus can be compiled in various ways. There are in fact various encodings, each one corresponding to a particular evaluation strategy in the λ -calculus. In particular, Milner showed that Plotkin's call-by-name and call-by-value strategies [35] can be both faithfully represented.

The way in which the representation is *faithful*, however, is quite subtle. It is looser than what one might expect, as the diagram in Figure 1.a *does not hold*. It is only possible to get the diagram in Figure 1.b: P_t , the process representing t , does not reduce to P_s , but to a process Q which is strongly bisimilar to P_s . One might think that a better encoding could solve this problem, but this is a naïve expectation: the two systems compute in radically different ways, the mismatch is inherent. In Milner's result P_s and Q are strongly bisimilar, which means that they behave the same *externally*, i.e. in their

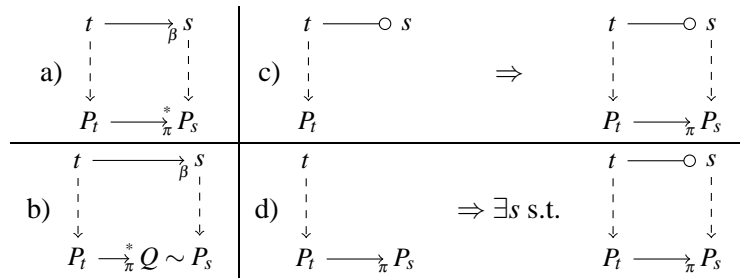


Figure 1: Diagrams describing the relationship between terms and processes.

interactions with every possible environment. However, the two processes behave in a quite different way *internally*, *i.e.* with respect to reductions. The discrepancy concerns the granularity of evaluation: λ -calculus uses a coarse, big-step substitution rule, while the π -calculus evaluates in small, fine-grained steps, as an abstract machine. Nonetheless, the evaluation of t terminates if and only if the evaluation of the corresponding process P_t terminates. In this sense, the representation is sometimes said to be sound and complete.

This paper refines the relationship between the λ -calculus and the π -calculus by extending the former with explicit substitutions—which may be considered as an alternative to abstract machines—in order to get a closer match of reduction steps. In the call-by-name case we show that the strategy corresponding to the evaluation in the π -calculus is exactly *linear weak head reduction* \multimap , the small-step head strategy of linear logic proof nets [29, 3]. This notion of evaluation has connections with Krivine’s abstract machine [20], Bohm’s separation theorem [29], computational complexity [9], the geometry of interaction [19], game semantics [18, 17], and the differential λ -calculus [24]. The relationship shown here is extremely strong. It is represented in the diagrams in Figure 1.c-d, which hold modulo structural equivalence only. They express the fact that the translation is a strong bisimulation *with respect to reduction* (note that *one* step maps to *one* step, and vice-versa).

The relationship between the π -calculus and linear logic has been analyzed from various points of view [31, 1, 12, 11, 27, 23, 15]. Our study essentially refines the work of Caires, Pfenning, and Toninho in [39], where the encodings of the λ -calculus in the π -calculus are re-understood as the encodings of λ -calculus into linear logic (due to Girard [26], see also [28]). The refinement consists in looking to such encodings via linear logic proof nets, but replacing the explicit use of proof nets with the lighter and equivalent reformulations as calculi of explicit substitutions *at a distance*, developed in [7, 8, 2, 10, 3, 5].

Contributions. In some sense there is not much original content in this paper. Damiano Mazza’s master thesis [30] (in French and unpublished) already developed the connection with linear weak head reduction. Similar ideas are sketched by Boudol in the introduction of [13]. Also, Milner’s seminal paper already suggested to use some environment device to refine the encodings, an idea that has then been explored by Vasconcelos [40] and recently by Cimini, Sacerdoti Coen, and Sangiorgi [16].

What is original here is the presentation. Our approach provides a remarkably compact development, confirming the relevance of explicit substitutions *at a distance* as a very flexible syntactical tool. Our presentation simplifies in the extreme Mazza’s study, by exploiting the simpler and more manageable reformulation of weak linear head reduction in the *linear substitution calculus* [9, 3]. In addition, by clarifying the connection with a crucial concept in the theory of linear logic, we get an important corollary *for free*. In [9] it is proven that linear head reduction is at most quadratically longer than head reduction, and this result holds also with respect to the weak (*i.e.* not under lambdas) variants of these reductions¹. Plotkin’s call-by-name strategy is the same thing as weak head reduction. Consequently, we get a quadratic relation between the call-by-name strategy and the evaluation in the π -calculus, which is a non-trivial quantitative refinement of Milner’s result.

However, our contribution is not only about the presentation. The study of call-by-name is complemented by the study of a call-by-value encoding, from which we extract a call-by-value \multimap_v analogous of linear weak head reduction, which has never been considered before. We also show that this new strategy enjoys the analogous of the *subterm property* [9] of linear weak head reduction, which is the basic property for complexity analysis. Last but not least, we give a presentation *at a distance* of the

¹The upper bound in [9] is exact, and it is based on a transformation of reductions which applies to arbitrary reduction sequences, in particular even to non-terminating terms. For instance, the quadratic bound is reached by the evaluation of $(\lambda x.xx)\lambda x.xx$, which is weak.

rewriting rules of the π -calculus which is a contribution of independent interest.

Despite the compactness of the presentation, the details turned out to be quite delicate. The use of *distance rules*, which are rewriting rules involving contexts (*i.e.* terms with holes), is crucial. They reflect on terms the local rules of linear logic proof nets, and they are essential in order to get a strong bisimulation of reductions. These contexts can capture variables and names, a fact which requires a very careful analysis of the translations. This is why we present the proofs of the translation in details, almost certifying the result. Moreover, we use colors to ease the reading, so we suggest to read the paper simultaneously on paper and on a computer screen.

The relationship with proof nets. Proof nets do not appear in this paper, we limit ourselves to the equivalent formulations as calculi at a distance. However, for the call-by-value calculus the detailed correspondence between terms and proof nets can be found in [5] (which uses big-step rules, while here we use small-step rules), for call-by-name the interested reader may have a look to [7, 2] (that do employ small-step rules, but in a slightly different way). On proof nets, linear head reduction is the small step strategy which reduces only the cuts at level 0 which do not involve the auxiliary conclusions of !-boxes. The weak variant can be defined in exactly the same way if boxes are also used for \wp (which in this context rather corresponds to the right rule for linear implication in intuitionistic linear logic, and not to the \wp of classical linear logic). Using boxes for linear implication is less *ad-hoc* than it may seem at first sight; a technical discussion of this issue is in Section 6 of [5]. This paper provides another justification for such boxes: they are needed to properly reflect evaluation in the π -calculus.

Plan of the paper. Section 1 introduces the linear substitution calculus, and Section 2 introduces the presentation of the π -calculus that we use. Sections 3 and 4 study the call-by-name and the call-by-value encodings, respectively.

Acknowledgements. To Frank Pfenning, for having encouraged me to work out the details of this work, and to Damiano Mazza, for inspiration and comments on an early draft. This work was partially supported by the Qatar National Research Fund under grant NPRP 09-1107-1-168.

1 The linear substitution calculus

The language of the *linear substitution calculus* λ_{sub} is given by the following grammar for terms:

$$t, s, u, r ::= x \mid \lambda x. t \mid ts \mid t[x/s]$$

The constructor $t[x/s]$ is called an *explicit substitution* (of s for x in t , the usual (implicit) substitution is instead noted $t\{x/s\}$). Both $\lambda x. t$ and $t[x/s]$ bind x in t . We are not going to define the full calculus (for which we refer to [9, 3]), but only linear weak head reduction. However, let us point out that the linear substitution calculus is a variation over a calculus of explicit substitutions introduced by Robin Milner in [34], to analyze the translation of λ -calculus to Bigraps.

We shall use contexts extensively, so we define them formally. In particular, we need to specify the set Δ of variables captured by a given context. A weak head context, or simply an **evaluation context**, is a term of the following grammar (to ease the reading on screen all contexts will be in blue):

$$E_0 ::= (\cdot) \mid E_0 t \qquad E_{\Delta \wp \{x\}} ::= E_{\Delta}[x/t] \mid E_{\Delta \wp \{x\}} t$$

A special case of evaluation context is given by **substitution contexts**, noted L_{Δ} and defined by:

$$L_0 ::= (\cdot) \qquad L_{\Delta \wp \{x\}} ::= L_{\Delta}[x/t]$$

Definition 1. Linear weak head reduction \multimap is defined as the union of \multimap_{dB} and \multimap_{1s} , which are given by the closure by evaluation contexts (*i.e.* $\multimap_{dB} := E_{\Delta}[\multimap_{dB}]$ and $\multimap_{1s} := E_{\Delta}[\multimap_{1s}]$) of the rules \mapsto_{dB} and \mapsto_{1s} defined as:

$$L_{\Delta}(\lambda x.t)s \mapsto_{\text{dB}} L_{\Delta}(t[x/s]) \qquad E_{\Delta}(x)[x/s] \mapsto_{\text{1s}} E_{\Delta}(s)[x/s] \quad \text{with } x \notin \Delta$$

The rule \mapsto_{1s} implicitly assumes the side-condition $\text{fv}(s) \cap \Delta = \emptyset$. The assumption is implicit because it can always be guaranteed by α -conversion: if $u = E_{\Delta}(x)[x/s]$ and $\text{fv}(s) \cap \Delta \neq \emptyset$ then there exist a set of variables Σ and an evaluation context F_{Σ} s.t. $u =_{\alpha} F_{\Sigma}(x)[x/s]$ and $\text{fv}(s) \cap \Sigma = \emptyset$.

These rule are *at a distance*, because their definition involves contexts, which is how locality on proof nets is reflected on terms. In Milner's calculus the first rule does not use $L_{\Delta}(\cdot)$. This is not a detail: the results in this paper would not hold with respect to Milner's original presentation.

It is natural to wonder in which sense the linear substitution calculus is *linear*. In contrast to other linear calculi, variables may have multiple occurrences, and arguments are not forced to be used only once. A first superficial linear aspect of the calculus is that variable occurrences are substituted one at the time. A second much deeper aspect is that its head strategy—characterized by a factorization theorem in the same way as head reduction in λ -calculus [3]—is *linear head reduction*, whose main feature is the *subterm property* (namely: any subterm u which is duplicated at any point of a reduction $t \rightarrow^k s$ is a subterm of t , whose size then does not depend on k) which implies that the implementation cost of *every* step is linear (in the size of t , the parameter for complexity). This is a fundamental property, not enjoyed by any strategy in λ -calculus (for which the cost of one step is not even polynomial in the size of t), and which opens the way to the study of computational complexity [9]. Here we deal with linear *weak* head reduction, which forbids reduction under abstractions. The restriction does not affect the subterm property.

2 The π -calculus

The fragment of the π -calculus we use here is essentially the asynchronous calculus in [21] with both unary and binary inputs and outputs, morally corresponding to the exponential and the multiplicative connectives of linear logic (in the typed case of [21]) and without sums (which correspond to the additives). The only change is that we do not use their forwarding processes². The grammar is:

$$P, Q, R ::= 0 \mid \bar{x}(y) \mid \bar{x}(y, z) \mid vxP \mid x(y, z).P \mid !x(y).P \mid P \mid Q$$

We need a notion of context also for processes. A **non-blocking context** is given by:

$$N_{\emptyset} ::= (\cdot) \mid N_{\emptyset} \mid Q \mid P \mid N_{\emptyset} \qquad N_{\Delta \oplus x} ::= vxN_{\Delta} \mid N_{\emptyset}(\lambda_{\Delta \oplus x})$$

The language is considered modulo **structural congruence**, *i.e.* the minimum equivalence relation generated by the following rules and closed by non-blocking contexts:

$$\begin{array}{c} \frac{}{P \mid 0 \equiv P} \quad \frac{}{P \mid (Q \mid R) \equiv (P \mid Q) \mid R} \quad \frac{}{P \mid Q \equiv Q \mid P} \\[10pt] \frac{}{vx0 \equiv 0} \quad \frac{x \notin \text{fn}(P)}{P \mid vxQ \equiv vx.(P \mid Q)} \quad \frac{}{vxvyP \equiv vyvxP} \end{array}$$

In order to prove the simulation theorems we will use the following three properties of \equiv , proved by easy inductions on N_{Δ} , P , and N_{Δ} , respectively (the set of free variables of a context is defined as for processes but using $\text{fn}((\cdot)) = \emptyset$).

Lemma 2. *Let Δ be a set of variables, N_{Δ} a non-blocking context, P a process s.t. $\text{fn}(P) \cap \Delta = \emptyset$, and $x, y \notin \Delta$. Then:*

²Forwarding processes correspond to axioms in linear logic. In terms of proof nets, avoiding forwarding processes correspond to use an interaction nets presentation, *i.e.* to work modulo cut-elimination on axioms.

1. $N_\Delta(Q) \mid P \equiv N_\Delta(Q \mid P)$.
2. If $x \notin \text{fn}(P)$ then $\nu x P \equiv P$.
3. If $x \notin \text{fn}(N_\Delta)$ then $\nu x N_\Delta(P) \equiv N_\Delta(\nu x P)$.

The rewriting rules are the following:

$$\bar{x}(y, z) \mid x(y', z').Q \rightarrow_\otimes Q\{y'/y\}\{z'/z\} \qquad \bar{x}(y) \mid !x(z).Q \rightarrow_! Q\{z/y\} \mid !x(z).Q$$

as usual they are both closed by non-blocking contexts and considered modulo \equiv . The second rule puts together replication and unary communication as in [39, 21].

π -calculus, at a distance. In order to simplify the proof of the bisimulation, we are going to use an alternative but equivalent definition of reduction in the π -calculus. Essentially, we have to reformulate the π -calculus *at a distance*. The use of the structural equivalence in the definition of the rewriting relation of the π -calculus induces some annoying complications when one tries to reflect process reductions on terms. We are going to reformulate the reduction rules via non-blocking contexts, and get rid of structural equivalence.

The rewriting rules \Rightarrow_\otimes and $\Rightarrow_!$ are given by the closure by non-blocking contexts (but are not closed by structural congruence) of the following relations: if $x \notin \Delta \cup \Gamma$ then

$$\begin{aligned} N_\Delta(\bar{x}(y, z)) \mid M_\Gamma(x(y', z').P) &\mapsto_\otimes M_\Gamma(N_\Delta(P\{y'/y\}\{z'/z\})) \\ N_\Delta(\bar{x}(y)) \mid M_\Gamma(!x(z).P) &\mapsto_! M_\Gamma(N_\Delta(P\{z/y\} \mid !x(z).P)) \end{aligned}$$

Actually, one should ask three further conditions on variables: 1) $\Delta \cap \Gamma = \emptyset$; 2) $\Delta \cap \text{fv}(P) = \emptyset$; 3) $\text{fv}(N_\Delta) \cap \Gamma = \emptyset$. It is easily seen, however, that these conditions can always be satisfied by choosing an α -equivalent term, as it is the case for the \mapsto_{1s} rule of λ_{lsub} . Essentially, these rules re-formulate as reduction rules the τ -transitions of the alternative presentation of the π -calculus as a labeled transition system, which is used to study the interaction of a process with its environment. Here, the new rules are more convenient than labeled transitions, because on λ -terms there is no analogous of the transitions whose label is not τ (and τ -transitions are defined using the non- τ transitions). This reformulation is justified by the following lemma, whose proof is along the one of the harmony lemma in [38] (p. 51).

Lemma 3.

1. \equiv is a strong bisimulation with respect to \Rightarrow : $P \equiv \Rightarrow_\otimes Q$ iff $P \Rightarrow_\otimes Q$, and $P \equiv \Rightarrow_! Q$ iff $P \Rightarrow_! Q$.
2. **Harmony of \Rightarrow and \rightarrow_π :** $P \rightarrow_\otimes Q$ iff $P \Rightarrow_\otimes Q$, and $P \rightarrow_! Q$ iff $P \Rightarrow_! Q$.

Curiously, the first formulation of the π -calculus was as a labeled transition system; the notions of reduction and structural congruence were introduced by Milner only later on, to study the relationship with the λ -calculus [33]. Our formulation at a distance of the π -calculus—motivated in exactly the same way—is a contribution of independent interest, probably the main one from the π -calculus point of view. It also shows that distance rules are a general syntactic principle whose relevance extends beyond explicit substitutions.

3 The call-by-name encoding

As for the ordinary λ -calculus, the translation from λ_{lsub} to the π -calculus is parametrized by a special channel name a . Actually, we assume that these **special channel names** are taken from a set A which is disjoint from the set of variable names, and whose elements are denoted a, b, c, d, \dots

The translation is given by (on screen it is in red):

$$\begin{aligned}
\llbracket x \rrbracket_a &:= \bar{x}\langle a \rangle & \llbracket ts \rrbracket_a &:= \nu b \nu x (\llbracket t \rrbracket_b \mid \bar{b}\langle x, a \rangle \mid !x(c). \llbracket s \rrbracket_c) \quad x \text{ is fresh} \\
\llbracket \lambda x. t \rrbracket_a &:= a(x, b). \llbracket t \rrbracket_b & \llbracket t[x/s] \rrbracket_a &:= \nu x (\llbracket t \rrbracket_a \mid !x(b). \llbracket s \rrbracket_b)
\end{aligned}$$

Modulo minor details, this is the original call-by-name encoding given by Milner. With respect to the relation with linear logic developed in [21], special names correspond exactly to multiplicative formulas, while variable names correspond to exponential formulas.

An easy induction on the translation shows:

Lemma 4. *Let t be a term. Then $\text{fn}(\llbracket t \rrbracket_a) = \text{fv}(t) \uplus \{a\}$.*

To relate terms and processes we need to prove a property of the translation, concerning its action on contexts: it maps evaluation contexts to non-guarding contexts of a special form.

Lemma 5 (Relating E and N via $\llbracket \cdot \rrbracket_a$). *Let Δ be a set of variable names, E_Δ an evaluation context, and a a special name. There exist a set of names Γ (possibly containing both variables and special names), a non-blocking context $N_{\Delta \uplus \Gamma}$ and a special name b s.t. $\llbracket E_\Delta(t) \rrbracket_a = N_{\Delta \uplus \Gamma}(\llbracket t \rrbracket_b)$ and $\Gamma \cap \text{fv}(t) = \emptyset$ for every term t . Moreover, if E_Δ is a substitution context L_Δ then $a = b$, $\Gamma = \emptyset$, and N_Δ does not depend on a .*

Proof. By induction on E_Δ . The base case is given by the empty context $E_0 = (\cdot)$, and it is trivial, just take $\Gamma := \emptyset$, $N_0 := (\cdot)$, and $b = a$. The inductive cases:

- *Left of an application, $E_\Delta = F_\Delta s$: if x is a fresh variable name:*

$$\begin{aligned}
\llbracket E_\Delta(t) \rrbracket_a &= \llbracket F_\Delta(t)s \rrbracket_a = \nu d \nu x (\llbracket F_\Delta(t) \rrbracket_d \mid \bar{d}\langle x, a \rangle \mid !x(c). \llbracket s \rrbracket_c) \\
&\stackrel{i.h.}{=} \nu d \nu x (M_{\Delta \uplus \Sigma}(\llbracket t \rrbracket_b) \mid \bar{d}\langle x, a \rangle \mid !x(c). \llbracket s \rrbracket_c) = N_{\Delta \uplus \Sigma \uplus \{d, x\}}(\llbracket t \rrbracket_b)
\end{aligned}$$

By *i.h.* we get that $\Sigma \cap \text{fv}(t) = \emptyset$. By definition of the translation x is fresh, so $x \notin \text{fv}(t)$. We then conclude by taking $\Gamma := \Sigma \uplus \{d, x\}$.

- *Left of a substitution, $E_{\Delta \uplus \{x\}} = F_\Delta[x/s]$:*

$$\begin{aligned}
\llbracket E_{\Delta \uplus \{x\}}(t) \rrbracket_a &= \llbracket F_\Delta(t)[x/s] \rrbracket_a = \nu x (\llbracket F_\Delta(t) \rrbracket_a \mid !x(c). \llbracket s \rrbracket_c) \\
&\stackrel{i.h.}{=} \nu x (M_{\Delta \uplus \Gamma}(\llbracket t \rrbracket_b) \mid !x(c). \llbracket s \rrbracket_c) = N_{\Delta \uplus \{x\} \uplus \Gamma}(\llbracket t \rrbracket_b)
\end{aligned}$$

and the *i.h.* also gives $\Gamma \cap \text{fv}(t) = \emptyset$.

Now suppose that $E_{\Delta \uplus \{x\}}$ (and thus F_Δ) is a substitution context L_Δ . Then by *i.h.* we get M_Δ not depending on a s.t.:

$$\begin{aligned}
\llbracket E_{\Delta \uplus \{x\}}(t) \rrbracket_a &= \llbracket F_\Delta(t)[x/s] \rrbracket_a = \nu x (\llbracket F_\Delta(t) \rrbracket_a \mid !x(c). \llbracket s \rrbracket_c) \\
&\stackrel{i.h.}{=} \nu x (M_\Delta(\llbracket t \rrbracket_a) \mid !x(c). \llbracket s \rrbracket_c) = N_{\Delta \uplus \{x\}}(\llbracket t \rrbracket_a)
\end{aligned}$$

Where clearly $N_{\Delta \uplus \{x\}}$ does not depend on a . □

We can now proceed with the simulation.

Theorem 6 (\rightarrow_π strongly simulates \multimap via $\llbracket \cdot \rrbracket_a$).

1. $t \multimap_{\text{dB}} s$ implies $\llbracket t \rrbracket_a \Rightarrow_{\otimes} \llbracket s \rrbracket_a$.
2. $t \multimap_{1s} s$ implies $\llbracket t \rrbracket_a \Rightarrow_{!} \llbracket s \rrbracket_a$.

Proof. 1. Two cases:

- *Root rewriting step:* first without $L_\Delta(\cdot)$: $(\lambda x. M)N \mapsto_{\text{dB}} M[x/N]$

$$\begin{aligned}
\llbracket (\lambda x.t)s \rrbracket_a &= \mathbf{v}b\mathbf{v}y(\llbracket \lambda x.t \rrbracket_b \mid \bar{b}\langle y, a \rangle \mid !y(c). \llbracket s \rrbracket_c) = \mathbf{v}b\mathbf{v}y(b(x, e). \llbracket t \rrbracket_e \mid \bar{b}\langle y, a \rangle \mid !y(c). \llbracket s \rrbracket_c) \\
&\Rightarrow_{\otimes} \mathbf{v}b\mathbf{v}y(\llbracket t \rrbracket_a \{x/y\} \mid !y(c). \llbracket s \rrbracket_c) =_{\alpha} \mathbf{v}b\mathbf{v}x(\llbracket t \rrbracket_a \mid !x(c). \llbracket s \rrbracket_c) \\
&= \mathbf{v}b\llbracket t[x/s] \rrbracket_a \equiv \llbracket t[x/s] \rrbracket_a
\end{aligned}$$

The $=_{\alpha}$ -step is justified by the fact that y is introduced fresh in the first line. The \equiv step is justified by Lemma 4, for which the only free special name occurring in $\llbracket t \rrbracket_a$ is a , and by Lemma 2.2, which allow us to remove the useless $\mathbf{v}b$.

Now, if $L_{\Delta}(\lambda x.t)s \mapsto_{\text{dB}} L_{\Delta}(t[x/s])$ we get (some explanations follow):

$$\begin{aligned}
\llbracket L_{\Delta}(\lambda x.t)s \rrbracket_a &= \mathbf{v}b\mathbf{v}y(\llbracket L_{\Delta}(\lambda x.t) \rrbracket_b \mid \bar{b}\langle y, a \rangle \mid !y(c). \llbracket s \rrbracket_c) \\
&=_{\text{Lem.5}} \mathbf{v}b\mathbf{v}y(N_{\Delta}(\llbracket \lambda x.t \rrbracket_b) \mid \bar{b}\langle y, a \rangle \mid !y(c). \llbracket s \rrbracket_c) \\
&= \mathbf{v}b\mathbf{v}y(N_{\Delta}(b(x, e). \llbracket t \rrbracket_e) \mid \bar{b}\langle y, a \rangle \mid !y(c). \llbracket s \rrbracket_c) \\
&\Rightarrow_{\otimes} \mathbf{v}b\mathbf{v}y(N_{\Delta}(\llbracket t \rrbracket_a \{x/y\} \{e/a\}) \mid !y(c). \llbracket s \rrbracket_c) \\
&=_{\alpha} \mathbf{v}b\mathbf{v}x(N_{\Delta}(\llbracket t \rrbracket_a) \mid !x(c). \llbracket s \rrbracket_c) \\
&\equiv_{\text{Lem.2.1 \& Lem.2.3}} \mathbf{v}bN_{\Delta}(\mathbf{v}x(\llbracket t \rrbracket_a \mid !x(c). \llbracket s \rrbracket_c)) \\
&= \mathbf{v}bN_{\Delta}(\llbracket t[x/s] \rrbracket_a) \\
&=_{\text{Lem.5}} \mathbf{v}b\llbracket L_{\Delta}(t[x/s]) \rrbracket_a \\
&\equiv_{\text{Lem.4 \& Lem.2.2}} \llbracket L_{\Delta}(t[x/s]) \rrbracket_a
\end{aligned}$$

The $=_{\alpha}$ -step and the last step are justified as before. In the first application of \equiv we can apply Lemma 2.1 because by hypothesis $x \notin \Delta$ and $\text{fv}(s) \cap \Delta = \emptyset$, and Lemma 2.3 because $x \notin \text{fn}(N_{\Delta})$. The two applications of Lemma 5 are with respect to different special names a and b , but this is sound: the *moreover* part of Lemma 5 guarantees that in the case of a substitution context L_{Δ} the corresponding context N_{Δ} does not depend on the name.

- *Inductive step:* $E_{\Delta}(t) \mapsto_{\text{dB}} E_{\Delta}(s)$ because $t \mapsto_{\text{dB}} s$. Let us recall that by definitions reductions in the π -calculus are closed by non-blocking contexts. Then:

$$\llbracket E_{\Delta}(t) \rrbracket_a =_{\text{Lem.5}} N_{\Delta \uplus \Gamma}(\llbracket t \rrbracket_b) \Rightarrow_{\otimes} N_{\Delta \uplus \Gamma}(\llbracket s \rrbracket_b) =_{\text{Lem.5}} \llbracket E_{\Delta}(s) \rrbracket_a$$

2. For \rightarrow_{1s} the inductive case is as for \rightarrow_{dB} . The base case is $E_{\Delta}(x)[x/s] \mapsto_{1s} E_{\Delta}(s)[x/s]$ with $x \notin \Delta$:

$$\begin{aligned}
\llbracket E_{\Delta}(x)[x/s] \rrbracket_a &= \mathbf{v}x(\llbracket E_{\Delta}(x) \rrbracket_a \mid !x(b). \llbracket s \rrbracket_b) =_{\text{Lem.5}} \mathbf{v}x(N_{\Delta \uplus \Gamma}(\llbracket x \rrbracket_c) \mid !x(b). \llbracket s \rrbracket_b) \\
&= \mathbf{v}x(N_{\Delta \uplus \Gamma}(\bar{x}\langle c \rangle) \mid !x(b). \llbracket s \rrbracket_b) \Rightarrow_{!} \mathbf{v}xN_{\Delta \uplus \Gamma}(\llbracket s \rrbracket_c \mid !x(b). \llbracket s \rrbracket_b) \\
&\equiv_{\text{Lem.2.1}} \mathbf{v}x(N_{\Delta \uplus \Gamma}(\llbracket s \rrbracket_c) \mid !x(b). \llbracket s \rrbracket_b) =_{\text{Lem.5}} \mathbf{v}x(\llbracket E_{\Delta}(s) \rrbracket_a \mid !x(b). \llbracket s \rrbracket_b) \\
&= \llbracket E_{\Delta}(s)[x/s] \rrbracket_a
\end{aligned}$$

where the \equiv -step is justified by the fact that by hypothesis and by Lemma 5 ($x \notin \Gamma$) we get that $(\text{fv}(s) \uplus \{x, b\}) \cap (\Delta \uplus \Gamma) = \emptyset$, and so we can apply Lemma 2.1. \square

The converse relation. To simulate process reductions on λ -terms we need a lemma, which is a converse to Lemma 5.

Lemma 7. *Let Δ and Γ be a set of variable names and a set of special names, respectively.*

1. *If $\llbracket t \rrbracket_a = N_{\Delta \uplus \Gamma}(a(y, b).P)$ with $a \notin \Gamma$ then $\Gamma = \emptyset$ and exist s and L_{Δ} s.t. $P = \llbracket s \rrbracket_b$ and $t = L_{\Delta}(\lambda y.s)$.*
2. *If $\llbracket t \rrbracket_a = N_{\Delta \uplus \Gamma}(\bar{x}\langle c \rangle)$ with $x \notin \Delta$ then exist $\Sigma \subseteq \Delta$ and E_{Σ} s.t. $t = E_{\Sigma}(x)$ (and $x \notin \Sigma$).*

Proof. Both points are by induction on t :

- *Variable:*

1. The hypothesis is false and there is nothing to prove.
 2. By definition of $\llbracket \cdot \rrbracket_a$, taking the empty context (and $\Delta = \emptyset$).
- *Abstraction*:
 1. By definition of $\llbracket \cdot \rrbracket_a$, taking the empty context (and $\Delta = \emptyset$).
 2. The hypothesis is false and there is nothing to prove.
 - *Application*: if $t = ur$ then $\llbracket ur \rrbracket_a = \nu b \nu z (\llbracket u \rrbracket_b \mid \bar{b}\langle z, a \rangle \mid !z(c). \llbracket r \rrbracket_c)$ with z fresh.
 1. By Lemma 4 $a \notin \text{fn}(\llbracket u \rrbracket_b)$, and so there is no context $N_{\Delta \uplus \Gamma}$ s. t. $\llbracket t \rrbracket_a = N_{\Delta \uplus \Gamma}(\langle a(y, b) \rangle.P)$, hence the hypothesis is false and there is nothing to prove.
 2. It must be that $\llbracket u \rrbracket_a = M_{\Delta' \uplus \Gamma'}(\bar{x}\langle c \rangle)$ with $\Delta = \Delta' \uplus \{z\}$ and $\Gamma = \Gamma' \uplus \{a\}$. Then by *i.h.* there exist $\Sigma \subseteq \Delta'$ and F_Σ s.t. $u = F_\Sigma(x)$. We conclude taking $E_\Sigma := F_\Sigma r$.
 - *Substitution*: if $t = u[z/r]$ then $\llbracket u[z/r] \rrbracket_a = \nu z (\llbracket u \rrbracket_a \mid !z(b). \llbracket r \rrbracket_b)$.
 1. If $\llbracket t \rrbracket_a = N_{\Delta \uplus \Gamma}(\langle a(y, b) \rangle.P)$ then it must be that exists $M_{\Delta' \uplus \Gamma'}(\cdot)$ with $\Delta = \Delta' \uplus \{z\}$ s.t. $\llbracket u \rrbracket_b = M_{\Delta' \uplus \Gamma'}(\langle a(y, b) \rangle.P)$ and $N_{\Delta \uplus \Gamma} = \nu z (M_{\Delta' \uplus \Gamma'} \mid !z(b). \llbracket r \rrbracket_b)$. By *i.h.* we get $\Gamma = \emptyset$, $u = L'_{\Delta'}(\lambda y. s)$, and $P = \llbracket s \rrbracket_b$. We conclude taking $L_\Delta := L'_{\Delta'}[z/r]$.
 2. It must be that $\llbracket u \rrbracket_a = M_{\Delta' \uplus \Gamma'}(\bar{x}\langle c \rangle)$ with $\Delta = \Delta' \uplus \{z\}$ and $\Gamma = \Gamma' \uplus \{a\}$. Then by *i.h.* there exist $\Sigma' \subseteq \Delta'$ and $F_{\Sigma'}$ s.t. $u = F_{\Sigma'}(x)$. We conclude taking $\Sigma := \Sigma' \uplus \{z\}$ and $E_\Sigma := F_{\Sigma'}[z/r]$. \square

Now, we can prove that any process reduction from $\llbracket t \rrbracket_a$ can be simulated by t .

Theorem 8 (\multimap strongly simulates \Rightarrow via $\llbracket \cdot \rrbracket_a$).

1. If $\llbracket t \rrbracket_a \Rightarrow_{\otimes} Q$ then exists s s.t. $t \multimap_{\text{dB}} s$ and $\llbracket s \rrbracket_a \equiv Q$.
2. If $\llbracket t \rrbracket_a \Rightarrow_! Q$ then exists s s.t. $t \multimap_{\text{ls}} s$ and $\llbracket s \rrbracket_a \equiv Q$.

Proof. Both points are by induction on t . Cases:

- *Values*: if $t = x$ or $t = \lambda x. u$ then $\llbracket t \rrbracket_a$ cannot reduce.
- *Application*: if $t = ur$ then $\llbracket t \rrbracket_a = \nu b \nu x (\llbracket u \rrbracket_b \mid \bar{b}\langle x, a \rangle \mid !x(c). \llbracket r \rrbracket_c)$ with x fresh. Then:
 1. *Multiplicative reduction*. Cases of $\llbracket t \rrbracket_a \Rightarrow_{\otimes} Q$:
 - *Root*: $\llbracket u \rrbracket_b = N_{\Delta \uplus \Gamma}(\langle b(y, d) \rangle.P)$ with $b \notin (\Delta \uplus \Gamma)$ and the process reduction is a \Rightarrow_{\otimes} interaction with $\bar{b}\langle x, a \rangle$ on b . By Lemma 7.1 we get that $\Gamma = \emptyset$, $u = L_\Delta(\lambda y. u')$, and $P = \llbracket u' \rrbracket_d$. So $t = L_\Delta(\lambda y. u')r$ and thus it has a \multimap_{dB} -redex on y , which maps to the \Rightarrow_{\otimes} communication on b exactly as in the proof of Theorem 6.1.
 - *Inductive*: because of $\llbracket u \rrbracket_b \Rightarrow_{\otimes} R$. Then by *i.h.* exists u' s.t. $u \rightarrow_{\text{dB}} u'$ and $\llbracket u' \rrbracket_b \equiv R$. We conclude by taking $s := u'r$.
 2. *Exponential reduction*. $\llbracket t \rrbracket_a \Rightarrow_! Q$ can only happen if reduction takes place in $\llbracket u \rrbracket_b$, because x is fresh by hypothesis. In such a case we conclude using the *i.h.*, as in the first sub-case of the previous point.
- *Substitution*: if $t = u[x/r]$ then $\llbracket t \rrbracket_a = \nu x (\llbracket u \rrbracket_a \mid !x(b). \llbracket r \rrbracket_b)$. We have:
 1. *Multiplicative reduction*. $\llbracket t \rrbracket_a \Rightarrow_{\otimes} Q$ can only happen if reduction takes place in $\llbracket u \rrbracket_a$, and we conclude using the *i.h.*.
 2. *Exponential reduction*. If $\llbracket t \rrbracket_a \Rightarrow_! Q$ because reduction takes place in $\llbracket u \rrbracket_a$ we use the *i.h.*. Otherwise, $\llbracket u \rrbracket_a = N_{\Delta \uplus \Gamma}(\bar{x}\langle c \rangle)$ with $x \notin \Delta \uplus \Gamma$ and the process reduction is a $\Rightarrow_!$ interaction with $!x(b). \llbracket r \rrbracket_b$ on x . By Lemma 7.2 there exist Σ and E_Σ s.t. $u = E_\Sigma(x)$. So $t = E_\Sigma(x)[x/r]$ has a \multimap_{ls} redex on x , which maps to the $\Rightarrow_!$ communication on x exactly as in the proof of Theorem 6.2. \square

According to the two theorems of this section, the relationship between the call-by-name strategy on the ordinary λ -calculus and the evaluation in the π -calculus is the same as the relationship between the call-by-name strategy and linear weak head reduction. In the strong case (*i.e.* when (head) reduction can go under lambdas), it is known that the latter can be at most quadratically longer than the former [9]. The analysis in [9] does not depend on being weak or strong. It follows that the same upper bound holds between the call-by-name strategy and its evaluation in the π -calculus.

Last, it is easy to see that linear weak head reduction is *deterministic*: every term has at most one \multimap redex, since every redex writes as $E_\Delta(\nu)$ (where ν is a value, *i.e.* a variable or an abstraction) and such a decomposition is unique. This property accounts for what Milner calls *determinacy* of $\llbracket t \rrbracket_a$ in [33].

4 The call-by-value encoding

We now show that the same exact relationship can be obtained with respect to call-by-value (CBV). The CBV calculus in use here is not Plotkin's calculus λ_{β_v} . In [10] the author and Paolini introduced the *value substitution calculus* λ_{vsub} , which is a CBV calculus with explicit substitutions containing λ_{β_v} as a sub-calculus and behaving better than λ_{β_v} with respect to the semantical notion of *solvability*. In [4, 5] we showed that λ_{vsub} has a sub-calculus, the *value substitution kernel* λ_{vker} , which has two key properties:

1. *Observational equivalence* [4]: there is a translation $\cdot^\circ : \lambda_{vsub} \rightarrow \lambda_{vker}$ s.t. t and t° are equivalent with respect to observing any termination property.
2. *Language for proof nets* [5]: λ_{vker} is an algebraic reformulation of the proof nets corresponding to the CBV translation of λ -calculus into linear logic. Namely, there is a translation $_ : \lambda_{vker} \rightarrow PN$ which is a strong bisimulation.

Here, we are going to show a further property: there are a CBV analogous \multimap_v of linear weak head reduction \multimap and a translation $\{\cdot\}^x$ from λ_{vker} to the π -calculus which is a strong bisimulation with respect to \multimap_v and \Rightarrow . Let us point out that in the untyped case there is a strong mismatch between Plotkin's calculus λ_{β_v} and the evaluation in proof nets (see [4]), thus the results of this section do not hold with respect to λ_{β_v} (nor with any of its refinements with explicit substitutions where β -redexes are constrained to fire on values).

The **value substitution kernel** λ_{vker} is given by the following grammar:

$$t, s, u, r ::= \nu \mid \nu t \mid t[x/s] \qquad \nu ::= x \mid \lambda x. t$$

Please note that the left sub-term of an application can only be a value (see [4, 5] for more details). Substitution contexts L_Δ are defined as before. Instead, the language of **evaluation contexts** changes:

$$E_0 ::= (\cdot) \mid \nu E_0 \mid t[x/E_0] \qquad E_{\Delta \uplus \{x\}} ::= E_\Delta[x/t] \mid \nu E_{\Delta \uplus \{x\}} \mid t[y/E_{\Delta \uplus \{x\}}]$$

Next, we define **applicative contexts** as $A_\Delta(\cdot) ::= E_\Delta((\cdot))$. As for CBN, we do not define the full calculus, but only the evaluation strategy. **Linear weak applicative reduction**, noted \multimap_v , is given by the rewriting rules $\multimap_{v\text{dB}}$ and $\multimap_{v\text{ls}}$ defined as the closure by evaluation contexts of the following rules:

$$(\lambda x. t)s \mapsto_{\text{dB}} t[x/s] \qquad A_\Delta(x)[x/L_\Sigma(\nu)] \mapsto_{\text{lsv}} L_\Sigma(A_\Delta(\nu)[x/\nu]) \quad x \notin \Delta$$

Note that the argument of a β -redex is not required to be a value, while the substitution rule can fire only in presence of a value (in a substitution context). As it was the case for the call-by-name calculus and for the π -calculus, one should also ask that $\text{fv}(\nu) \cap \Delta = \emptyset$, $\text{fv}(A_\Delta(x)) \cap \Sigma = \emptyset$, and $\Delta \cap \Sigma = \emptyset$, but these side-conditions can always be satisfied by taking an α -equivalent term, and so in the following they will be taken for granted. Note that $x[x/y] \not\mapsto_{\text{lsv}} y$ but $(xz)[x/y] \mapsto_{\text{lsv}} yz$, because substitution has to take place

in an applicative context. This applicative restriction is a sort of converse to the head restriction used in the case of call-by-name evaluation. In terms of proof nets both these restrictions correspond to forbid reduction of cuts involving links in some $!$ -boxes (with respect to the respective encodings of CBV and CBN), while the *weak* requirement correspond to the analogous constraint with respect to the \mathfrak{A} -boxes mentioned in the introduction. The applicative restriction is somehow a surprise, which is justified by the fact that it matches what happens in the π -calculus. It is a quite reasonable restriction: there is no point in substituting a value if it cannot be used in some application.

Linear weak applicative reduction enjoys a property which is the CBV analogous of the subterm property (defined at the end of Section 1). Let us call a *v-subterm* a subterm which is a value.

Lemma 9 (v-subterm property). *If $t \multimap_v^k s$ and v is a v-subterm of s then v is a v-subterm of t .*

Proof. By induction on k . For $k = 0$ it is trivial, for $k > 0$ consider the term u s.t. $u \multimap_v s$. The $\multimap_{v\text{dB}}$ rule does not create new values. The $\multimap_{v\text{LS}}$ rule duplicates a v-subterm of u , which by *i.h.* is a v-subterm of t , and it does not substitute into v-subterms. So, any v-subterm of s is a v-subterm of t . \square

Differently from linear weak head reduction, linear weak applicative reduction is a *non-deterministic* strategy: just consider $t = ((\lambda x.x)(yy))[y/z]$, which has two redexes. However, a simple induction shows that reduction is confluent: there is no need to use parallel reductions or other sophisticated techniques because no redex can duplicate/erase other redexes. In fact, it is easily seen that linear weak applicative reduction enjoys the diamond property. This fact corresponds to what Milner calls *determinacy* of the CBV encoding.

The translation. Similarly to the CBV translation of the λ -calculus to linear logic, the CBV translation to the π -calculus uses an auxiliary function. The main translation function $\{\!\{t\}\!\}^x$ is parametrized by a variable name $x \notin \text{fv}(t)$ (and not by a special name) and the auxiliary function is noted $\{\!\{\cdot\}\!\}^a$, *i.e.* we use the same symbol but now the parameter is a special name a :

$$\begin{aligned} \{\!\{v\}\!\}^x &::= !x(a). \{\!\{v\}\!\}^a & \{\!\{vs\}\!\}^x &::= v b v y (\{\!\{v\}\!\}^b \mid \overline{b}\langle y, x \rangle \mid \{\!\{s\}\!\}^y) \quad y \text{ is fresh} \\ \{\!\{y\}\!\}^a &::= \overline{y}\langle a \rangle & \{\!\{s[y/u]\}\!\}^x &::= v y (\{\!\{s\}\!\}^x \mid \{\!\{u\}\!\}^y) \\ \{\!\{\lambda y.s\}\!\}^a &::= a(y, z). \{\!\{s\}\!\}^z \end{aligned}$$

Note that the application case uses the auxiliary function on v . Note also the difference with the call-by-name case: applications and explicit substitutions do not use replication, which is instead associated to values, with the important exception of applied values. The *applicative* restriction on the strategy \multimap_v comes from this exception: the impossibility of interacting under replication in the π -calculus reflects on terms as the fact that one can substitute only on variables in applicative contexts, because the others are under a replication prefix. Last, this encoding is a minor variation over the CBV one in [39], which is not Milner's original CBV encoding.

Lemma 10. *Let $t \in \lambda_{\text{vker}}$. Then $\text{fn}(\{\!\{t\}\!\}^x) = \text{fv}(t) \uplus \{x\}$ and $\text{fn}(\{\!\{t\}\!\}^a) = \text{fv}(t) \uplus \{a\}$.*

Proof. By mutual induction on $\{\!\{t\}\!\}^x$ and $\{\!\{t\}\!\}^a$. \square

The following lemma is the call-by-value analogous of Lemma 5.

Lemma 11 (Relating E and N via $\{\!\{\cdot\}\!\}^x$). *Let Δ be a set of variable names, x a variable name and E_Δ an evaluation context. There exist a set of names Γ (possibly containing both variables and special names), a non-blocking context $N_{\Delta \uplus \Gamma}$, and a variable name z s.t. $\{\!\{E_\Delta(t)\}\!\}^x = N_{\Delta \uplus \Gamma}(\{\!\{t\}\!\}^z)$ and $\Gamma \cap \text{fv}(t) = \emptyset$ for every term t . Moreover, if E_Δ is a substitution context L_Δ then $x = z$, $\Gamma = \emptyset$, and N_Δ does not depend on x .*

Proof. By induction on E_Δ . The base case is given by the empty context $E_0 = (\cdot)$, and it is trivial, just take $\Gamma := \emptyset$, $N_0 := (\cdot)$, and $z := x$. The inductive cases:

- *Right of an application*, $E_\Delta = vF_\Delta$:

$$\begin{aligned} \{E_\Delta(t)\}^x &= \{vF_\Delta(t)\}^x = vbvy(\{v\}^b \mid \bar{b}\langle y, x \rangle \mid \{F_\Delta(t)\}^y) \\ &=_{i.h.} vbvy(\{v\}^b \mid \bar{b}\langle y, x \rangle \mid M_{\Delta \uplus \Sigma}(\{t\}^z)) = N_{\Delta \uplus \Sigma \uplus \{y, b\}}(\{t\}^z) \end{aligned}$$

The *i.h.* also gives $\Sigma \cap \text{fv}(t) = \emptyset$. Since $b, y \notin \text{fv}(t)$ it follows that $\Gamma := \Sigma \uplus \{y, b\}$ satisfies $\Gamma \cap \text{fv}(t) = \emptyset$.

- *Right of a substitution*, $E_\Delta = s[y/F_\Delta]$:

$$\begin{aligned} \{E_\Delta(t)\}^x &= \{s[y/F_\Delta(t)]\}^x = vy(\{s\}^x \mid \{F_\Delta(t)\}^y) \\ &=_{i.h.} vy(\{s\}^x \mid M_{\Delta \uplus \Sigma}(\{t\}^z)) = N_{\Delta \uplus \Sigma \uplus \{y\}}(\{t\}^z) \end{aligned}$$

The *i.h.* also gives $\Sigma \cap \text{fv}(t) = \emptyset$. Since $y \notin \text{fv}(t)$ it follows that $\Gamma := \Sigma \uplus \{y\}$ satisfies $\Gamma \cap \text{fv}(t) = \emptyset$.

- *Left of a substitution*, $E_{\Delta \uplus \{z\}} = F_\Delta[y/u]$. Then:

$$\begin{aligned} \{E_{\Delta \uplus \{z\}}(t)\}^x &= \{F_\Delta(t)[y/u]\}^x = vy(\{F_\Delta(t)\}^x \mid \{u\}^y) \\ &=_{i.h.} vy(M_{\Delta \uplus \Gamma}(\{t\}^z) \mid \{u\}^y) = N_{\Delta \uplus \{y\} \uplus \Gamma}(\{t\}^z) \end{aligned}$$

The *i.h.* also gives $\Gamma \cap \text{fv}(t) = \emptyset$. Now, suppose that $E_{\Delta \uplus \{y\}}$ (and thus F_Δ) is a substitution context L_Δ . Then by *i.h.* we get M_Δ not depending on x s.t.:

$$\begin{aligned} \{E_{\Delta \uplus \{y\}}(t)\}^x &= \{F_\Delta(t)[y/u]\}^x = vy(\{F_\Delta(t)\}^x \mid \{u\}^y) \\ &=_{i.h.} vy(M_\Delta(\{t\}^x) \mid \{u\}^y) = N_{\Delta \uplus \{y\}}(\{t\}^x) \end{aligned}$$

where clearly $N_{\Delta \uplus \{y\}}$ does not depend on x . \square

Theorem 12 (\rightarrow_π strongly simulates \rightarrow_{\circ_v}).

1. $t \rightarrow_{\text{vdb}} s$ implies $\{t\}^x \Rightarrow_{\otimes} \{s\}^x$.
2. $t \rightarrow_{\text{vls}} s$ implies $\{t\}^x \Rightarrow_{\dagger} \{s\}^x$.

Proof. We show the base cases, the inductive ones are as in the call-by-name case, using Lemma 11.

1. If $(\lambda y.t)s \rightarrow_{\text{vdb}} t[y/s]$ then:

$$\begin{aligned} \{(\lambda y.t)s\}^x &= vbvz(\{\lambda y.t\}^b \mid \bar{b}\langle z, x \rangle \mid \{s\}^z) = vbvy(b(y, w). \{t\}^w \mid \bar{b}\langle z, x \rangle \mid \{s\}^z) \\ &\Rightarrow_{\otimes} vbvy(\{t\}^w \{w/x\} \{y/z\} \mid \{s\}^z) =_{\alpha} vbvy(\{t\}^x \mid \{s\}^y) \\ &= vb\{t[x/s]\}^x \equiv_{\text{Lem.10}} \{t[x/s]\}^x \end{aligned}$$

2. If $A_\Delta(y)[y/L_\Sigma(v)] \mapsto_{\text{lsv}} L_\Sigma(A_\Delta(v)[y/v])$ and $A_\Delta(\cdot) = E_\Delta((\cdot)s)$ then:

$$\begin{aligned} \{A_\Delta(y)[y/L_\Sigma(v)]\}^x &= vy(\{E_\Delta(y)s\}^x \mid \{L_\Sigma(v)\}^y) \\ &=_{\text{Lem.11}} vy(N_{\Delta \uplus \Gamma}(\{ys\}^z) \mid M_\Sigma(\{v\}^y)) \\ &= vy(N_{\Delta \uplus \Gamma}(\{ys\}^z) \mid M_\Sigma(!y(a). \{v\}^a)) \\ &= vy(N_{\Delta \uplus \Gamma}(vbvw(\{y\}^b \mid \bar{b}\langle w, z \rangle \mid \{s\}^w)) \mid M_\Sigma(!y(a). \{v\}^a)) \\ &= vy(N_{\Delta \uplus \Gamma}(vbvw(\bar{y}\langle b \rangle \mid \bar{b}\langle w, z \rangle \mid \{s\}^w)) \mid M_\Sigma(!y(a). \{v\}^a)) \\ &\Rightarrow_{\dagger} vyM_\Sigma(N_{\Delta \uplus \Gamma}(vbvw(\{v\}^b \mid !y(a). \{v\}^a \mid \bar{b}\langle w, z \rangle \mid \{s\}^w))) \\ &\equiv_{\text{Lem.2.1}} vyM_\Sigma(N_{\Delta \uplus \Gamma}(vbvw(\{v\}^b \mid \bar{b}\langle w, z \rangle \mid \{s\}^w)) \mid !y(a). \{v\}^a) \\ &= vyM_\Sigma(N_{\Delta \uplus \Gamma}(\{vs\}^z) \mid !y(a). \{v\}^a) \\ &= vyM_\Sigma(\{E_\Delta(vs)\}^x \mid !y(a). \{v\}^a) \\ &\equiv_{\text{Lem.2.3}} M_\Sigma(vy(\{E_\Delta(vs)\}^x \mid !y(a). \{v\}^a)) \\ &= M_\Sigma(\{E_\Delta(vs)[y/v]\}^x) \\ &=_{\text{Lem.11}} \{L_\Sigma(E_\Delta(vs)[y/v])\}^x \\ &= \{L_\Sigma(A_\Delta(v)[y/v])\}^x \end{aligned}$$

The \equiv step after the reduction is justified by the fact that b , w , and all the variables in Γ are introduced fresh and so do not belong to $\text{fv}(v)$. Moreover, $\Delta \cap \text{fv}(v) = \emptyset$ by hypothesis, and so we can apply Lemma 2.1. \square

The converse relation. As for call-by-name, we show that linear weak applicative reduction reflects exactly evaluation in the π -calculus.

Lemma 13. *Let Δ and Γ be a set of variable names and a set of special names, respectively. Then:*

1. *If $\{t\}^x = N_{\Delta \uplus \Gamma}(!x(a).P)$ with $x \notin \Delta$ then $\Gamma = \emptyset$ and exist v and L_Δ s.t. $P = \{v\}^a$ and $t = L_\Delta(v)$.*
2. *If $\{t\}^x = N_{\Delta \uplus \Gamma}(\bar{y}(a))$ with $y \notin \Delta$ then exist $\Sigma \subseteq \Delta$ and A_Σ s.t. $t = A_\Sigma(y)$.*

Proof. Both points are by induction on t :

- *Value:* if $t = v'$ then $\{t\}^x = !x(a). \{v'\}^a$.
 1. Clearly $\Gamma = \Delta = \emptyset$, v is v' , and L_Δ is the empty context.
 2. The hypothesis is false, and so there is nothing to prove.
- *Application:* if $t = v's$ then $\{v's\}^x = v b v z(\{v'\}^b \mid \bar{b}(z, x) \mid \{s\}^z)$ with z and b are fresh.
 1. By definition of the translation $x \notin \text{fv}(v's)$ and so by Lemma 10 $x \notin \text{fn}(\{v'\}^b) \cup \text{fn}(\{s\}^z)$. Consequently, there is no context $N_{\Delta \uplus \Gamma}$ s. t. $\{t\}^x = N_{\Delta \uplus \Gamma}(!x(a).P)$, so the hypothesis is false and there is nothing to prove.
 2. Two cases:
 - (a) $\{v'\}^b = \bar{y}(a)$ and $N_{\Delta \uplus \Gamma} = v b v z(\{v'\}^b \mid \bar{b}(z, x) \mid \{s\}^z)$, which imply $v' = y$, $a = b$, $\Delta = \{z\}$, and $\Gamma = \{b\}$. We conclude taking $\Sigma := \emptyset$ and $A_0 := (\cdot)_s$.
 - (b) The context hole $(\cdot)_s$ is in $\{s\}^z$. Let $\Delta' := \Delta \setminus \{z\}$ and $\Gamma' := \Gamma \setminus \{b\}$. If $\{t\}^x = N_{\Delta \uplus \Gamma}(\bar{z}(a))$ then $\{s\}^x = M_{\Delta' \uplus \Gamma'}(\bar{z}(a))$ for some context $M_{\Delta' \uplus \Gamma'}$. The *i.h.* gives $\Sigma \subseteq \Delta'$ and an applicative context B_Σ s.t. $s = B_\Sigma(y)$. We conclude taking $A_\Sigma := v' B_\Sigma$.
- *Substitution:* if $t = s[z/u]$ then $\{t\}^x = v z(\{s\}^x \mid \{u\}^z)$.
 1. By definition of the translation $x \notin \text{fv}(s[z/u])$ and so by Lemma 10 $x \in \text{fn}(\{s\}^x)$ and $x \notin \text{fn}(\{u\}^z)$. Consequently, the context hole $(\cdot)_s$ is in $\{s\}^x$, which then writes as $M_{\Delta' \uplus \Gamma'}(!x(a).P)$, with $\Delta = \Delta' \uplus \{z\}$ for some context $M_{\Delta' \uplus \Gamma'}$. By *i.h.* we get that there exist v and $L'_{\Delta'}$ s.t. $P = \{v\}^a$ and $s = L'_{\Delta'}(v)$. We conclude taking $L_\Delta := L'_{\Delta'}[z/u]$.
 2. Two cases:
 - (a) The context hole $(\cdot)_s$ is in $\{s\}^x$. Let $\Delta' := \Delta \setminus \{z\}$. If $\{t\}^x = N_{\Delta \uplus \Gamma}(\bar{z}(a))$ then $\{s\}^x = M_{\Delta' \uplus \Gamma'}(\bar{z}(a))$ for some context $M_{\Delta' \uplus \Gamma'}$. The *i.h.* gives $\Sigma' \subseteq \Delta'$ and an applicative context $B_{\Sigma'}$ s.t. $s = B_{\Sigma'}(y)$. We conclude taking $\Sigma := \Sigma' \uplus \{z\}$ and $A_\Sigma := B_{\Sigma'}[z/u]$.
 - (b) The context hole is in $\{u\}^z$. Analogous to the previous case (except that $\Sigma = \Sigma'$). \square

Theorem 14 (\dashv_{v} strongly simulates \Rightarrow via $\{ \cdot \}^a$).

1. *If $\{t\}^x \Rightarrow_{\otimes} Q$ then exists r s.t. $t \dashv_{\text{vdB}} r$ and $\{r\}^x \equiv Q$.*
2. *If $\{t\}^x \Rightarrow_{\text{!}} Q$ then exists r s.t. $t \dashv_{\text{vls}} r$ and $\{r\}^x \equiv Q$.*

Proof. By induction on t . Cases:

- *Values:* if t is a value then $\{t\}^x$ cannot reduce.
- *Application:* if $t = vs$ then $\{vs\}^x = v b v y(\{v\}^b \mid \bar{b}(y, x) \mid \{s\}^y)$ with y and b fresh. Then:

1. *Multiplicative reduction.* Cases of $\llbracket t \rrbracket_x \Rightarrow_{\otimes} Q$:
 - *Root:* $\llbracket v \rrbracket^b = b(z, w).P$ interacts with $\bar{b}\langle y, x \rangle$ on b . Clearly, v is an abstraction $\lambda z.u$ with $\llbracket u \rrbracket^w = P$, and $t = (\lambda z.u)s$ has a root \multimap_{vdb} redex. Then, t and $\llbracket t \rrbracket^x$ are related exactly as in the proof of Theorem 12.1. Note that $b \notin \text{fn}(\llbracket s \rrbracket^y)$ by Lemma 10, and so there cannot be any multiplicative root interaction involving $\llbracket s \rrbracket^y$.
 - *Inductive:* $\llbracket t \rrbracket^x \Rightarrow_{\otimes} Q$ because $\llbracket s \rrbracket^y \Rightarrow_{\otimes} P$. By *i.h.* we get that there exists r' s.t. $s \rightarrow_{\text{dB}} r'$ and $\llbracket r' \rrbracket^y \equiv P$. Since $v(\cdot)$ is an evaluation contexts, taking $r := vr'$ we get $t \rightarrow_{\text{dB}} r$ and $\llbracket r \rrbracket^x \equiv P$.
 2. *Exponential reduction.* The inductive case (*i.e.* $\llbracket t \rrbracket^x \Rightarrow_{!} Q$ because $\llbracket s \rrbracket^y \Rightarrow_{\otimes} P$) follows by the *i.h.* as in the inductive case for multiplicative reductions. In the root case there cannot be any root exponential reduction. Indeed, $\llbracket v \rrbracket^b$ would have to be $\bar{z}\langle b \rangle$ and $\llbracket s \rrbracket^y$ should have a $!z(c).P$ sub-process. This second requirement is only possible if s contains a value v which in $\llbracket s \rrbracket^y$ is translated with respect to z , so that $\llbracket v \rrbracket^z = !z(c).P$. But this is impossible because y is fresh (and so $y \neq z$) and any variable name which is used as a parameter in the translation of a subterm of s is either y or it is introduced fresh (and so cannot be equal to z).
- *Substitution:* if $t = \llbracket s[y/u] \rrbracket^x$ then $\llbracket t \rrbracket^x = \text{vy}(\llbracket s \rrbracket^x \mid \llbracket u \rrbracket^y)$
 1. *Multiplicative reduction.* If the reduction takes place in $\llbracket s \rrbracket^x$ or $\llbracket u \rrbracket^y$ we use the *i.h.* as in the previous inductive cases. And there cannot be any root multiplicative reduction. Indeed, it should be along a special name a free in both $\llbracket s \rrbracket^x$ and $\llbracket u \rrbracket^y$, but by Lemma 10 $\llbracket s \rrbracket^x$ and $\llbracket u \rrbracket^y$ have no free special name.
 2. *Exponential reduction.* If the reduction takes place in $\llbracket s \rrbracket^x$ or $\llbracket u \rrbracket^y$ we use the *i.h.* as in the previous inductive cases.
 Otherwise, an exponential reduction can only be along a variable name z which is free in both $\llbracket s \rrbracket^x$ and $\llbracket u \rrbracket^y$. Then $z \neq x$, because $x \notin \text{fn}(\llbracket u \rrbracket^y)$. Another requirement is that z has to be used as the parameter of the translation of a value v , which is the only way to get a replicated input. The only possibility then is that $z = y$, because all variable parameter names used in the translation and different from x and y are fresh and cannot be in both $\llbracket s \rrbracket^x$ and $\llbracket u \rrbracket^y$.
 Now, $\llbracket s \rrbracket^x$ has to be of the form $N_{\Delta \uplus \Gamma}(\bar{y}\langle a \rangle)$ and $\llbracket u \rrbracket^y$ has to be of the form $M_{\Delta' \uplus \Gamma'}(!y(b).P)$, for some sets of variable names Δ and Δ' and some sets of special names Γ and Γ' , and with $y \notin \Delta \cup \Delta'$. By Lemma 13 we get $\Gamma' = \emptyset$ and that exist v , $L_{\Delta'}$, $\Sigma \subset \Delta$, and A_{Σ} s.t. $P = \llbracket v \rrbracket^b$, $u = L_{\Delta'}(v)$, and $s = A_{\Sigma}(y)$. Summing up, $t = A_{\Sigma}(y)[y/L_{\Delta'}(v)]$ and it has a \multimap_{vls} redex which maps on $\llbracket t \rrbracket_x \Rightarrow_{!} Q$ exactly as in the proof of Theorem 12.2. \square

Conclusions

We have shown how to refine the relation between the λ -calculus and the π -calculus, getting a perfect match of reductions steps in both call-by-name and call-by-value. The refinements crucially exploits rewriting rules at a distance, and unveil that the π -calculus evaluates λ -terms exactly as linear logic proof nets. A natural continuation would be to extend these relations to calculi with multiplicities [14], which are related to the study of observational equivalence. It would also be interesting to investigate linear weak applicative reduction, in particular in relation with complexity [9] or with Taylor-Ehrhard expansion [22]. Finally, given the compactness of the results and the involved reasoning about bound, free, and fresh variables, it would be interesting to try to formalize this work in Abella [25], which is a proof assistant provided with a nominal quantifier precisely developed to cope with the π -calculus [32] and where reasoning about untyped calculi with binders is very close to pen-and-paper reasoning [6].

References

- [1] Samson Abramsky (1993): *Computational Interpretations of Linear Logic*. *Theor. Comput. Sci.* 111(1&2), pp. 3–57. Available at [http://dx.doi.org/10.1016/0304-3975\(93\)90181-R](http://dx.doi.org/10.1016/0304-3975(93)90181-R).
- [2] Beniamino Accattoli (2011): *Jumping around the box: graphical and operational studies on λ -calculus and Linear Logic*. PhD thesis, La Sapienza University of Rome.
- [3] Beniamino Accattoli (2012): *An Abstract Factorization Theorem for Explicit Substitutions*. In: *RTA*, pp. 6–21. Available at <http://dx.doi.org/10.4230/LIPIcs.RTA.2012.6>.
- [4] Beniamino Accattoli (2012): *A linear analysis of call-by-value λ -calculus*. Available at the address <https://sites.google.com/site/beniaminoaccattoli/Accattoli-Alinearanalysisofcall-by-valuelambdac>.
- [5] Beniamino Accattoli (2012): *Proof nets and the call-by-value λ -calculus*. *LSFA 2012*. Available at the address <https://sites.google.com/site/beniaminoaccattoli/Accattoli-Proofnetsandthecallbyvaluelambdac>.
- [6] Beniamino Accattoli (2012): *Proof Pearl: Abella Formalization of λ -Calculus Cube Property*. In: *CPP*, pp. 173–187. Available at http://dx.doi.org/10.1007/978-3-642-35308-6_15.
- [7] Beniamino Accattoli & Stefano Guerrini (2009): *Jumping Boxes*. In: *CSL*, pp. 55–70. Available at http://dx.doi.org/10.1007/978-3-642-04027-6_7.
- [8] Beniamino Accattoli & Delia Kesner (2010): *The Structural λ -Calculus*. In: *CSL*, pp. 381–395. Available at http://dx.doi.org/10.1007/978-3-642-15205-4_30.
- [9] Beniamino Accattoli & Ugo Dal Lago (2012): *On the Invariance of the Unitary Cost Model for Head Reduction*. In: *RTA*, pp. 22–37. Available at <http://dx.doi.org/10.4230/LIPIcs.RTA.2012.22>.
- [10] Beniamino Accattoli & Luca Paolini (2012): *Call-by-Value Solvability, revisited*. In: *FLOPS*, pp. 4–16. Available at http://dx.doi.org/10.1007/978-3-642-29822-6_4.
- [11] Emmanuel Beffara (2006): *A Concurrent Model for Linear Logic*. *Electr. Notes Theor. Comput. Sci.* 155, pp. 147–168. Available at <http://dx.doi.org/10.1016/j.entcs.2005.11.055>.
- [12] Gianluigi Bellin & Philip J. Scott (1994): *On the π -Calculus and Linear Logic*. *Theor. Comput. Sci.* 135(1), pp. 11–65. Available at [http://dx.doi.org/10.1016/0304-3975\(94\)00104-9](http://dx.doi.org/10.1016/0304-3975(94)00104-9).
- [13] Gérard Boudol (1998): *The π -Calculus in Direct Style*. *Higher-Order and Symbolic Computation* 11(2), pp. 177–208. Available at <http://dx.doi.org/10.1023/A:1010064516533>.
- [14] Gérard Boudol & Cosimo Laneve (1996): *The Discriminating Power of Multiplicities in the Lambda-Calculus*. *Inf. Comput.* 126(1), pp. 83–102. Available at <http://dx.doi.org/10.1006/inco.1996.0037>.
- [15] Luís Caires & Frank Pfenning (2010): *Session Types as Intuitionistic Linear Propositions*. In: *CONCUR*, pp. 222–236. Available at http://dx.doi.org/10.1007/978-3-642-15375-4_16.
- [16] Matteo Cimini, Claudio Sacerdoti Coen & Davide Sangiorgi (2010): *Functions as Processes: Termination and the $\lambda\mu\tilde{\mu}$ -Calculus*. In: *TGC*, pp. 73–86. Available at http://dx.doi.org/10.1007/978-3-642-15640-3_5.
- [17] Pierre Clairambault (2011): *Estimation of the Length of Interactions in Arena Game Semantics*. In: *FOS-SACS*, pp. 335–349. Available at http://dx.doi.org/10.1007/978-3-642-19805-2_23.
- [18] Vincent Danos, Hugo Herbelin & Laurent Regnier (1996): *Game Semantics & Abstract Machines*. In: *LICS*, pp. 394–405. Available at <http://doi.ieeecomputersociety.org/10.1109/LICS.1996.561456>.
- [19] Vincent Danos & Laurent Regnier (1999): *Reversible, Irreversible and Optimal lambda-Machines*. *Theor. Comput. Sci.* 227(1-2), pp. 79–97. Available at [http://dx.doi.org/10.1016/S0304-3975\(99\)00049-3](http://dx.doi.org/10.1016/S0304-3975(99)00049-3).
- [20] Vincent Danos & Laurent Regnier (2004): *Head Linear Reduction*. Technical Report.
- [21] Henry DeYoung, Luís Caires, Frank Pfenning & Bernardo Toninho (2012): *Cut Reduction in Linear Logic as Asynchronous Session-Typed Communication*. In: *CSL*, pp. 228–242. Available at <http://dx.doi.org/10.4230/LIPIcs.CSL.2012.228>.

- [22] Thomas Ehrhard (2012): *Collapsing non-idempotent intersection types*. In: *CSL*, pp. 259–273. Available at <http://dx.doi.org/10.4230/LIPIcs.CSL.2012.259>.
- [23] Thomas Ehrhard & Olivier Laurent (2010): *Interpreting a finitary pi-calculus in differential interaction nets*. *Inf. Comput.* 208(6), pp. 606–633. Available at <http://dx.doi.org/10.1016/j.ic.2009.06.005>.
- [24] Thomas Ehrhard & Laurent Regnier (2006): *Böhm Trees, Krivine’s Machine and the Taylor Expansion of Lambda-Terms*. In: *CiE*, pp. 186–197. Available at http://dx.doi.org/10.1007/11780342_20.
- [25] Andrew Gacek (2008): *The Abella Interactive Theorem Prover (System Description)*. In: *IJCAR*, pp. 154–161. Available at http://dx.doi.org/10.1007/978-3-540-71070-7_13.
- [26] Jean-Yves Girard (1987): *Linear Logic*. *Theoretical Computer Science* 50, pp. 1–102. Available at [http://dx.doi.org/10.1016/0304-3975\(87\)90045-4](http://dx.doi.org/10.1016/0304-3975(87)90045-4).
- [27] Kohei Honda & Olivier Laurent (2010): *An exact correspondence between a typed pi-calculus and polarised proof-nets*. *Theor. Comput. Sci.* 411(22-24), pp. 2223–2238. Available at <http://dx.doi.org/10.1016/j.tcs.2010.01.028>.
- [28] John Maraist, Martin Odersky, David N. Turner & Philip Wadler (1999): *Call-by-name, Call-by-value, Call-by-need and the Linear lambda Calculus*. *Theor. Comput. Sci.* 228(1-2), pp. 175–210. Available at [http://dx.doi.org/10.1016/S0304-3975\(98\)00358-2](http://dx.doi.org/10.1016/S0304-3975(98)00358-2).
- [29] Gianfranco Mascari & Marco Pedicini (1994): *Head Linear Reduction and Pure Proof Net Extraction*. *Theor. Comput. Sci.* 135(1), pp. 111–137. Available at [http://dx.doi.org/10.1016/0304-3975\(94\)90263-1](http://dx.doi.org/10.1016/0304-3975(94)90263-1).
- [30] Damiano Mazza (2003): *Pi et Lambda. Une étude sur la traduction des lambda-termes dans le pi-calcul*. Memoire de DEA (in french).
- [31] Dale Miller (1992): *The pi-Calculus as a Theory in Linear Logic: Preliminary Results*. In: *ELP*, pp. 242–264. Available at http://dx.doi.org/10.1007/3-540-56454-3_13.
- [32] Dale Miller & Alwen Tiu (2010): *Proof search specifications of bisimulation and modal logics for the π -calculus*. *ACM Trans. Comput. Log.* 11(2). Available at <http://doi.acm.org/10.1145/1656242.1656248>.
- [33] Robin Milner (1992): *Functions as Processes*. *Math. Str. in Comput. Sci.* 2(2), pp. 119–141. Available at <http://dx.doi.org/10.1017/S0960129500001407>.
- [34] Robin Milner (2007): *Local Bigraphs and Confluence: Two Conjectures*. *Electr. Notes Theor. Comput. Sci.* 175(3), pp. 65–73. Available at <http://dx.doi.org/10.1016/j.entcs.2006.07.035>.
- [35] Gordon D. Plotkin (1975): *Call-by-Name, Call-by-Value and the lambda-Calculus*. *Theor. Comput. Sci.* 1(2), pp. 125–159. Available at [http://dx.doi.org/10.1016/0304-3975\(75\)90017-1](http://dx.doi.org/10.1016/0304-3975(75)90017-1).
- [36] Davide Sangiorgi (1994): *The Lazy Lambda Calculus in a Concurrency Scenario*. *Inf. Comput.* 111(1), pp. 120–153. Available at <http://dx.doi.org/10.1006/inco.1994.1042>.
- [37] Davide Sangiorgi (1999): *From lambda to pi; or, Rediscovering continuations*. *Math. Str. in Comput. Sci.* 9(4), pp. 367–401. Available at <http://dx.doi.org/10.1017/S0960129599002881>.
- [38] Davide Sangiorgi & David Walker (2001): *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press.
- [39] Bernardo Toninho, Luís Caires & Frank Pfenning (2012): *Functions as Session-Typed Processes*. In: *FoS-SaCS*, pp. 346–360. Available at http://dx.doi.org/10.1007/978-3-642-28729-9_23.
- [40] Vasco Thudichum Vasconcelos (2005): *Lambda and pi calculi, CAM and SECD machines*. *J. Funct. Program.* 15(1), pp. 101–127. Available at <http://dx.doi.org/10.1017/S0956796804005386>.